Prototipo Avanzado de un Simulador de Planificación de Tareas de Tiempo Real en Sistemas Heterogéneos

Martín Joel Rodríguez [1] y Claudio Aciti [1 2]

[1] Universidad Nacional de Tres de Febrero, Sede Caseros
Valentín Gómez 4828, Caseros (B1678ABJ), Buenos Aires, Argentina

[2] Universidad Nacional del Centro de la Provincia de Buenos Aires.
Pinto 399, Tandil (7000), Buenos Aires, Argentina

{mjrodriguez@untref.edu.ar, caciti@exa.unicen.edu.ar}

Abstract. Se desarrolló una aplicación que simula el comportamiento de un conjunto de tareas periódicas en sistemas de tiempo real con procesadores heterogéneos. El simulador permite elegir entre diferentes algoritmos de planificación, se implementaron PF y EDF, y es extensible a incorporar nuevos algoritmos. Para los casos de inclumplimiento de plazos, se implementaron los mecanismos Aborto y Finalización Tardía, y es extensible a incorporar nuevos mecanismos. En cada simulación, se debe definir previamente un conjunto de datos con las configuraciones de los procesadores y las tareas que serán simuladas, y se le debe especificar el número de iteraciones a evaluar. El simulador procesa esta información y entrega el resultado.

Keywords: Sistemas de tiempo real - Procesadores heterogéneos - Planificación de tareas

1 Introducción

Los sistemas heterogéneos avanzan paulatinamente ofreciendo altos niveles de performance y de optimización del uso de energía, comparados con los sistemas tradicionales [1]. Históricamente, los sistemas de tiempo-real (STR) se ejecutaban en sistemas monoprocesadores, sin embargo, en la actualidad los STR están migrando a arquitecturas heterogéneas, es decir, arquitecturas de computación que cuentan con múltiples elementos de procesamiento para proporcionar un balance óptimo entre rendimiento, latencia, flexibilidad, costo, entre otros factores [2, 3].

1.1 Antecedentes

Los sistemas heterogéneos, están formados por distintos procesadores pudiendo contener uno o más de cada tipo. Estos sistemas, pueden utilizar diferentes tipos de procesadores, por ejemplo, un procesador de propósito general (GPP) y un procesador

de propósito especial (GPU, FPGA, DSP, etc.) [4]. Dentro de un sistema heterogéneo es posible implementar un STR. Por definición, un STR, es aquel encargado de producir respuestas dentro de un determinado tiempo finito, es decir, la calidad de las respuestas del sistema se ve condicionada por el momento en que se producen las mismas [5, 6].

Uno de los principales retos que surge es la planificación de las tareas de tiempo-real en multiprocesadores heterogéneos. Los STR, por lo general, están constituidos por tareas periódicas que incluyen, entre sus parámetros, los instantes máximos en que las mismas deben finalizar su ejecución. Este parámetro extra se denomina deadline[1]. Si una tarea finaliza después de este tiempo, se dice que ha perdido su vencimiento [4, 7].

Los STR, por lo general, cuentan con un Sistema Operativo de Tiempo Real (RTOS) donde una de las principales funciones es la planificación de las tareas a ejecutar. El planificador debe elegir cual tarea ejecutar a continuación. Para hacer esto, primero, examina la cola de tareas listas para su ejecución y, dependiendo de la política de ejecución implementada, elige una para ejecutarla. Al planificar tareas en sistemas heterogéneos, la complejidad aumenta notablemente, porque no solo hay que decidir en qué procesador se va a ejecutar la tarea siguiente, sino que además el RTOS debe decidir en qué tipo de procesador se puede realizar la ejecución. Es por esto que, de antemano, el RTOS debe saber los tipos de procesadores de los cuales dispone y en cuales se puede ejecutar cada una de las tareas que integran el sistema [8].

En la actualidad, un conjunto de aplicaciones y frameworks existen en la literatura para la simulación de planificaciones de tareas en STR. Entre los relevados se pueden mencionar a STRESS [9], PERTS [10], YASA [11], Cheddar [12], RealTTS [13] y el simulador de la Université Libre de Bruxelles [14]. Aplicaciones como MAST [15] ofrecen herramientas de modelado, y otras como FORTISSIMO [16] presentan una plataforma para el diseño de simulaciones.

1.2 Motivación y Objetivo

La atracción generada en la última década por los sistemas heterogéneos produjo que el desarrollo de nuevas estrategias de planificación sea un activo campo de investigación debido a que los algoritmos de planificación juegan un importante papel en los sistemas heterogéneos [17]. Para explotar todo el potencial de un componente heterogéneo es necesario asignar correctamente la carga de trabajo. La planificación de tareas y el equilibrio de carga del sistema se vuelven más complejos en los sistemas heterogéneos. En los STR, la planificación de las tareas es fundamental para garantizar que las tareas se ejecuten correctamente dentro de los tiempos previamente asignados.

Considerando este contexto y la problemática de la planificación, que resulta fundamental en todo sistema, junto con el déficit generado por no encontrar un software capaz de realizar planificaciones de tareas en STR con procesadores

[1] Del inglés: vencimiento.

heterogéneos, se propone como objetivo el desarrollo e implementación de un prototipo de simulador de planificador de tareas en STR con procesadores heterogéneos.

Entre los objetivos parciales, el prototipo de simulador deberá permitir seleccionar el algoritmo de planificación y el mecanismo para el manejo de tareas no cumplidas por sobrecarga. El simulador realizará planificaciones de *N* tareas en *M* procesadores (pudiendo estos ser distintos), y a cada tarea se le podrá indicar en qué tipo de procesador podrá ejecutarse, ya que al trabajar con sistemas heterogéneos puede darse el caso de que una tarea sea ejecutable en más de un tipo de procesador o por lo contrario que una tarea pueda ejecutarse en una tipo de procesador y no en otro.

Se propone modularizar cada paso de la utilización de la herramienta, en aplicaciones virtualmente independientes, que pueden ser programadas en un lenguaje a elección, implementando una comunicación entre estos módulos, lo que permita al usuario reemplazar cualquier módulo por uno de su propio diseño, u obtener los datos en crudo de la simulación si así lo desea.

2 Definición formal del problema

Un STR consiste en un conjunto *P* de *N* tareas periódicas. Cada tarea $\tau_i \in \Pi$ está caracterizada por su período, T_i, el tiempo límite para ejecutarse, D_i, el tiempo de ejecución, C_i, el tiempo de inicio (offset o jitter), f_i y una prioridad, P_i. La unidad mínima de cómputo de una tarea T_i es una acción *J*. Una tarea es un conjunto de acciones similares que se repiten a lo largo del tiempo.

$$\Pi = \{\tau_i = (T_i, D_i, C_i, f_i, P_i)\} \quad (1)$$

Un sistema con multiprocesadores heterogéneos está compuesto por un conjunto Φ de tipos de procesadores diferentes φ_x.

$$\Phi = \{\varphi_{x,y},\ 1 \le x \le M,\ y \ge 0\} \quad (2)$$

El subíndice *"x"* indica el tipo de procesador y el subíndice *"y*" indica la cantidad.

2.1. STR de tareas periódicas en procesadores heterogéneos

Un STR de tareas periódicas con procesadores heterogéneos está compuesto por:

- Un conjunto de tareas $\tau_i \in \Pi$.
- Un conjunto de procesadores $\varphi_{x,y} \in \Phi$.
- Una lista de procesadores y tareas definida por pares ordenados $<\varphi_{x,y}, \tau_i>$

y sigue las condiciones que se especifican a continuación:

- Cada tarea τ_i tiene asignada una prioridad. La tarea que tiene mayor frecuencia es la que tiene mayor prioridad. Y la que tiene menor frecuencia es la que tiene menor prioridad.
- Una tarea τ_i puede ser ejecutada en uno o más tipos de procesadores $\varphi_{x,y}$.
- Una vez que una instancia q representada por τ_i, q de una tarea τ_i inició su ejecución, sólo puede continuar en procesadores del mismo tipo.
- Cada tarea deberá tener una definición para cada tipo de procesador en la que pueda ser ejecutada.

2.2. Paso a paso de la Planificación de tareas en un procesador

Los pasos propuestos para planificar una tarea en un sistema heterogéneo son:

A. El planificador chequea, en cada instante de tiempo *t*, si una tarea τ_i inicia un período nuevo (una instancia nueva) en la estructura *T*. En ese caso, $\tau_{i,q}$ debería agregarse al conjunto de tareas *LISTAS (L)*. Esto significa que la instancia anterior $\tau_{i,\,q-1}$ llegó a su tiempo de finalización.
B. Si la instancia $\tau_{i,\,q-1}$ completó su ejecución. Entonces $\tau_{i,\,q}$ es agregada al conjunto de tareas *LISTAS (L)*.
C. Si la instancia $\tau_{i,\,q-1}$ empezó su ejecución pero no la completó su ejecución. Entonces $\tau_{i,\,q-1}$ sigue a la espera en el conjunto de tareas *LISTAS (L)* y $\tau_{i,\,q-1}$ es descartada.
D. El planificador debe decidir qué tarea τ_i se va a ejecutar en el instante *t* siguiente.
E. Si no hay ninguna tarea en ejecución busca en las tareas *LISTAS* la de mayor prioridad y la ejecuta.
F. Si hay una tarea en ejecución, compara con la de mayor prioridad de las tarea *LISTAS*. Si la que está en ejecución tiene mayor prioridad o igual, entonces sigue. Caso contrario realiza un cambio de contexto.
G. Ejecuta la tarea.
H. Vuelve al punto A.

3 Solución propuesta

Se puede considerar la aplicación que realiza la simulación como la caja negra central, y las entradas y salidas de la aplicación como módulos separados independientes. Llamaremos a la aplicación que realiza la simulación backend, mientras que los módulos de entrada y salida serán parte del frontend, que serán las partes que tendrán interacción con el usuario. El módulo de entrada es el "Panel de definiciones" donde el usuario designará los procesadores y tareas que serán simulados. El módulo de salida constituye el "Gráfico de la simulación", y es donde el usuario podrá visualizar el resultado obtenido, analizar eventos y sacar conclusiones.

3.1 Lenguaje de comunicación

Al tratarse de módulos separados e independientes, se define una manera de transmitir las entradas, en un lenguaje que pueda interpretar, al backend del simulador. De la misma manera, la salida del backend debe poder ser interpretada por cualquier módulo que se quiera implementar, sin quedar restringido al funcionamiento del mismo.

Se opta por el lenguaje JSON (JavaScript Object Notation) para la intercomunicación entre módulos independientes. JSON es un formato de estándar abierto que usa texto legible por los seres humanos para transmitir datos de objetos que consisten en pares atributo-valor, muy utilizado para la comunicación asincrónica entre un servidor y el explorador.

De esta forma se puede implementar los módulos de entrada y salida en cualquier lenguaje, siempre y cuando la entrada pueda expresar la información en JSON y la salida pueda interpretar el JSON para realizar la visualización de datos.

3.2 Backend

El backend es el motor principal de la aplicación, ya que es el encargado de hacer la simulación. Recibe una entrada con la definición de procesadores y tareas que están involucrados y otros datos como el planificador de tareas que se utilizará y la cantidad de tiempo (en número de iteraciones) que se quiere simular.

Es implementado en el lenguaje Java principalmente por ser orientado a objetos, por ser multiplataforma y por tener un buen manejo de la memoria (Java cuenta con un garbage collector, o recolector de basura, para deshacerse de los objetos instanciados que ya no son accedidos). El simulador maneja muchas instancias de tareas que luego de ejecutadas son desalojadas de los procesadores y no son vueltas a utilizar.

La aplicación Java fue montada como una aplicación web en un servidor Tomcat. Esto permite que pueda ser accedida por un explorador web, enviando y recibiendo datos a través de peticiones HTTP. De esta forma, los módulos de entrada y salida

pueden comunicarse al backend del simulador estando virtualmente alojado en cualquier servidor.

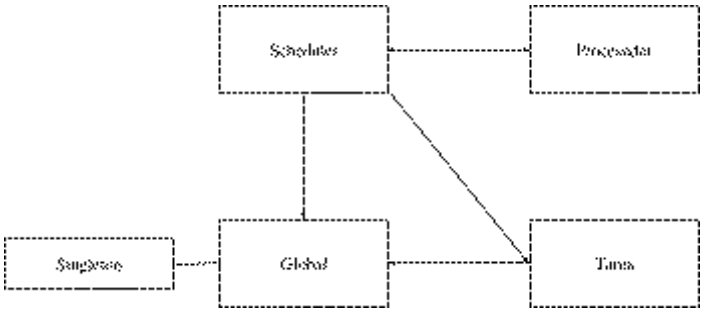

Fig 1: Diagrama de clases

Una primera aproximación es modelar los procesadores como si fueran procesadores homogéneos, es decir, que cada procesador se comporte de la misma forma y compute a la misma velocidad. En contraste, los procesadores heterogéneos sólo pueden recibir tareas específicas para un tipo de procesador, y su velocidad de cómputo puede variar según el caso. En este modelo cada procesador realiza una unidad de cómputo por unidad de tiempo.

Implementación de procesadores heterogéneos. Una vez implementada esta clase se modificó la arquitectura de la aplicación para hacer posible la existencia de procesadores heterogéneos, es decir, procesadores que solo pueden computar tareas de un tipo determinado. A su vez, los tipos de procesadores pueden diferir en su velocidad de procesamiento, haciendo que algunos computen más rápido que otros.

En principio se definieron las clases *Procesador* y *Tarea* como clases abstractas, de manera que no puedan instanciarse. Obligatoriamente los tipos de procesadores y tareas nuevas deben heredar de estas clases respectivamente.

Se define un atributo clock, cuyo valor por defecto es 1, que representa cuántas unidades de tiempo se necesitan para completar un cómputo de tarea. De esta manera, el valor 1 representa la velocidad más rápida, comportamiento esperado en la versión con procesadores homogéneos.

Para crear un tipo de procesador en particular se debe heredar de la clase Procesador. En su constructor se pasa además de su id, su velocidad de clock.

Es responsabilidad de la clase hijo sobreescribir el método *setTarea()* para restringir el tipo de tareas que puede recibir el nuevo tipo de procesador.

3.3 Frontend

El frontend constituye tanto el módulo de entrada de datos, como el de salida y visualización de la simulación. Al tratarse de módulos separados del backend de simulación, existen infinitas implementaciones que pueden ser compatibles con el simulador, siempre y cuando el lenguaje de intercomunicación sea JSON y la especificación de la transmisión de datos sea la acordada.

Para este trabajo se implementa un módulo de entrada y uno de salida como parte de una posible solución. La ventaja de tener el simulador modularizado de forma independiente es que pueden existir múltiples alternativas para las aplicaciones que se conecten al simulador.

3.4 Gráfico de la simulación

El gráfico es la última instancia de todo el recorrido de la aplicación y representa la visualización de la simulación. Al igual que el panel de definiciones, el gráfico de la simulación de este proyecto es sólo una de las infinitas soluciones que se pueden construir gracias a la modularidad del proyecto (Fig 2).

Si bien el gráfico es la salida del backend, se puede considerar a su vez como una segunda caja negra que recibe los datos de la simulación, los ordena y convierte en información gráfica para consumo del usuario. Esta transformación de los datos no es trivial y requiere algunas funciones que empaqueten los datos en formas que el usuario pueda comprender.

Representaciones. Existen varias formas de representar la simulación a partir de los datos recibidos. Debido a que se ponen en observación dos elementos distintos del sistema, procesadores y tareas, el gráfico puede representarse en función de uno o el otro. Se puede hacer una línea por cada procesador y cada casillero ocuparlo con la representación de la tarea en cuestión, o por el contrario, una línea por cada tarea y cada casillero ocuparlo con la representación del procesador en donde esta tarea se encuentra alojada. Por último se puede tener una vista cruzada, es decir, una línea por cada relación tarea-procesador posible.

Distintas vistas permiten llegar a una conclusión más exhaustiva de la simulación observada. Es por eso que para esta solución se implementaron tres visualizaciones distintas: Vista por Procesador, Vista por Tarea y Vista Cruzada.

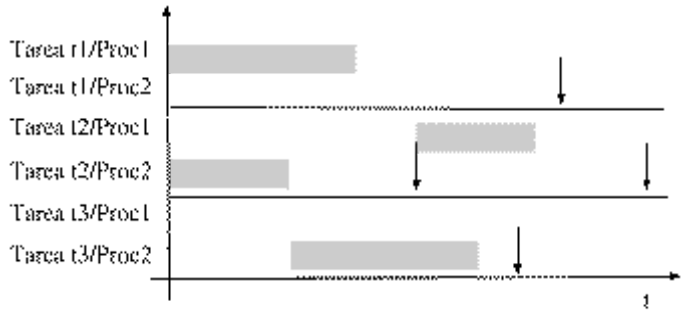

Fig. 2. Ejemplo de graficación en vista cruzada.

4 Conclusiones

El objetivo del presente trabajo ha sido desarrollar un prototipo de simulador de planificación de tareas de tiempo-real con multiprocesadores heterogéneos. Este objetivo ha sido alcanzado en su totalidad mediante el desarrollo de una solución modularizada de implementación independiente para el frontend y el backend, permitiendo una gran flexibilidad y futura explotación del simulador.

El backend de simulación se llevó a cabo como una aplicación Java alojada en un servidor web, y el frontend con HTML/CSS/JavaScript para poder armar las simulaciones y visualizarlas. Se incluye en el desarrollo la implementación de un planificador de prioridades fijas, con y sin desalojo configurable, y políticas de aborto para las fallidas. El simulador maneja tareas periódicas independientes con distintas definiciones de tiempo de inicio, cómputo, prioridad y plazo máximo de ejecución; simuladas en un grupo de procesadores heterogéneos con tipo y velocidad de cómputo configurable.

Se distinguieron y estudiaron los puntos críticos del planificador para los cuales se documentaron casos testigos para comprobar el correcto comportamiento del simulador en dichas circunstancias. Como resultado se observó que el simulador respondía de la forma esperada.

En términos generales, el simulador y la solución de frontend propuesta, brinda un entorno de estudio y desarrollo altamente expandible para futuros trabajos y soluciones alternativas, orientadas al estudio y planificación de STR.

5 Trabajos Futuros

Se destacan como trabajos futuros en la aplicación: Incorporar la posibilidad de simular tareas aperiódicas y esporádicas, definiendo una función de probabilidad para la aparición de estas, y habilitando la posibilidad de que puedan ejecutarse una única

vez. Dar acceso al simulador a través de un servidor web, aprovechando las ventajas del procesamiento dedicado y acceso remoto

Referencias

1. Brodtkorb, A. R., Dyken, C., Hagen, T. R., Hjelmervik, J. M., and Storaasli, O. O. (2010).
   State-of-the-art in heterogeneous computing. Scientific Programming, 18(1):1–33.
2. Luna, F., Nesmachnow, S., y Alba, E. (2010). Búsqueda local paralela para planificación de tareas en sistemas de computación heterogéneos. Universidad de Castilla, La Mancha
3. Orozco, J. D., Urriza, J. M., Cayssials, R., Fernández, E., Ferrari, M., Echaiz, J., Buckle, C., Barry, D., Páez, F. E., Olguín, G., et al. (2013). Sistemas de tiempo real con requerimientos heterogéneos: integración hardware-software. In XV Workshop de Investigadores en Ciencias de la Computación
4. Laplante, P. A. (1992). Real-Time Systems Design and Analysis: An Engineer's Handbook. IEEE Press, Piscataway, NJ, USA
5. Liu, C. L. and Layland, J. W. (1973). Scheduling Algorithm for multiprogramming in Hard-Real-Time Environment. Jornal of ACM.
6. Burns, A. y Wellings, A. J. (2010). Real-time systems and programming languages, volume 2097. Addison-Wesley
7. Urriza, J. , Ferrari, M., Orozco, J., Cayssials, R., Páez, F., Olguín, G., Schorb, L., Lucas, S., Tolosa, R., (2016). Sistemas de Tiempo Real Mixtos: Planificación en Sistemas Operativos de Tiempo Real Bajo Plataformas de Desarrollo Concretas. XVIII Workshop de Investigadores en Ciencias de la Computación (WICC 2016, Entre Ríos, Argentina)
8. Aguilar, J., Leiss, E., et al. (2004). Introducción a la computación paralela. Editorial Venezolana, Universidad de Los Andes, Mérida.
9. Audsley, N. C., Burns, A., Richardson, M. F., and Wellings, A. J. (1994). Stress: A simulator
   for hard real-time systems. Software: Practice and Experience, 24(6):543–564.
10. Liu, J. W.-S., Liu, C. L., Deng, Z., Tia, T.-S., Sun, J., Storch, M., Hull, D., Redondo, J., Bettati, R., and Silberman, A. (1996). Perts: A prototyping environment for real-tim e systems. International Journal of Software Engineering and Knowledge Engineering, 6(02):161– 177.
11. Golatowski, F., Hildebrandt, J., Blumenthal, J., and Timmermann, D. (2002). Framework for validation, test and analysis of real-time scheduling algorithms and scheduler implementations. In Rapid System Prototyping, 2002. Proceedings. 13th IEEE International Workshop on pages 146–152. IEEE.
12. Singhoff, F., Legrand, J., Nana, L., and Marcé, L. (2004). Cheddar: a flexible real time scheduling framework. In ACM SIGAda Ada Letters, volume 24, pages 1–8. ACM.
13. Diaz, A., Batista, R., and Castro, O. (2007). Realtss: a real-time scheduling simulator. In
    Electrical and Electronics Engineering, 2007. ICEEE 2007. 4th International

Conference
on, pages 165–168. IEEE.
14. Vroey, S. D., Goossens, J., and Hernalsteen, C. (1996). A generic simulator of real-time scheduling algorithms. In Simulation Symposium, 1996., Proceedings of the 29th Annual, pages 242–249. IEEE.
15. González Harbour, M., García, J. G., Gutiérrez, J. P., and Moyano, J. D. (2001). Mast: Modeling and analysis suite for real time applications. In Real-Time Systems, 13th Euromicro Conference on, 2001., pages 125–134. IEEE.
16. Kramp, T., Adrian, M., and Koster, R. (2000). An open framework for real-time scheduling
simulation. Springer.
17. Mednis, A., Strazdins, G., Zviedris, R., Kanonirs, G., and Selavo, L. (2011). Real time pothole detection using android smartphones with accelerometers. In Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on, pages 1–6. IEEE.